APPENDIX – A

SYNTAX AND SEMANTICS OF RCP STATEMENTS:

The syntax and semantics of the Rcp Statements, are presented below :

Syntactical Rules :

All Rcp statements begin with Exec Rcp token, and end with End_exec token. The Exec Rcp token is followed by the Rcp Statement name. Each Rcp statement contains a fixed number of parameters. Each parameter has a name, and the developer specifies a value for the parameter. Rcp Statements are not case sensitive, and upper and lower case characters are treated as same; however parameters are passed as is, and the interpretation is left to the host language.

The parms are specified as : PARM_NAME (Parm_value).

Statements can be specified on multiple lines, and can be broken whenever a keyword (token) ends;

For example, the parameter Num_of_elem can be specified on two lines as:

Num_of_elem
(10)

If the parameter value includes spaces or special characters it should be placed inside double quotes;

For example : Name (" This is a valid parameter value")

The actual pram value in this case is : This is a valid parameter value;

Please note the absence of quotes in the final value.

If the parameter value itself has double quotes then the escape sequence \" should be used to represent the double quotes.

For example : Name (" \" This is a quoted parameter value \" ")

The actual parameter value in this case is -> "This is a quoted parameter value";

Please note the presence of double quotes around the parameter value.

Classification of Rcp Statements :

Rcp statements are classified according to their usage :

1) Statements which can be specified in the .Resource definition file
2) Statements which can be specified in the program files
3) Statements which can be specified in both the .Resource definition file and the program files

It may be noted that executable Rcp Statements specified in the .Resource definition file will find support in Rcp_Init.cpp, which is automatically generated by the Rcp translator.

Semantic Rules :

Rcp statements can take variables or constants as parms. There are a few exceptions to this rule, and they are specified below :

1) Variable names and symbols defined to Rcp are indistinguishable. This problem is resolved by placing a '$' infront of the parameter value to inform that it is a host variable name.

For example, in the parameter Name (Q1), Q1 is a queue name which Rcp will try to translate, by looking into its symbol tables, whereas

In the parameter Name ($q2), q2 is regarded as a host language variable name and is passed to the host language compiler.

2) The symbol '#' is reserved for future use, and cannot be used as the first character of a variable name.

Rcp Error Check :

The Rcp Translator will generate code, which checks the return code of the Rcp library function corresponding to the Rcp Statement, and terminates further processing when the Rcp Statement fails. The developer can change this behavior and check return codes for specific failures by specifying the parameter Error_Check (YES) in the statement.

It may be noted that Abends cannot be bypassed, and the Rcp runtime will abend as soon as the node function returns control to the Rcp Runtime.

Rcp Statements :

1) Exec RCP [Resource definition file/Program file]
   Set Stmt_Num
   Psp (value)
   Cps (value)
   End_exec

This statement is used to change the statement numbers that are generated by

the program.

2) Exec RCP [Resource definition file/Program file]

Set Tab_Size

Count (value)

End_exec

This statement can be used to inform Rcp about the tab size of the source editor, so that the translated statements are indented correctly in the translated program.

3) Exec RCP [Resource definition file]

Create Frames

Count (value)

End_exec

This statement is used to create frames in the program.

4) Exec RCP [Resource definition file]

Create Threads

Min (value)

Max (value)

Frame (frame_num)

End_exec

This statement is used to create threads in the frame, note that Frame_num is optional, and if it is not specified the same number of threads are created in all the frames.

5) Exec RCP [Resource definition file]

Define Rcp_Gate

Name (Rcp_gate_name)

Qlist1 (queue_list_1)

Qlist2 (queue_list_2)

End_exec

This statement creates a Rcp_Gate and specifies the inputs/outputs of the Rcp_Gate. Either Qlist1 or Qlist2 parms has to be specified on this statement.

6) Exec RCP [Resource definition file]

Define Fctn

Name (fctn_name)

Qlist1 (queue_list_1)

Qlist2 (queue_list_2)

Rcp_Gate (rcp_gate_name)

End_exec

This statement creates a Rcp Node function and specifies the inputs/outputs of the function.

7) Exec RCP [Program file]

Run Pgm

Mode (security_mode)

End_exec

This statement starts the execution of the Rcp runtime library.The MODE parameter specifies the security mode in which the program will be executed.

8) Exec RCP [Program file]

Stop Run

End_exec

This statement will request Rcp to Stop the Frame. Frame will stop only after all functions have terminated.

9) Exec RCP [Program file]

Term Run

End_exec

This statement will terminate a Frame. All executing workers are terminated, immediately, and the frame is stopped immediately.

10) Exec RCP [Program file]

Term Pgm

End_exec

This statement will terminate all frames within the program.

11) Exec RCP [Resource definition file]

Define Queue

Name (queue_name)

Type (type)

Disp (queue_name)

End_exec

This statement will define a queue to the Rcp Implementation. It may be noted that only the name and type are specified at this time, the remaining details are captured from the Create statement.

12) Exec RCP [Resource definition file/Program file]

Create Queue

Name (queue_name)

Elem_size (value)

Num_of_elem (value)

Status (Ready / Not_Ready)

Error_Check (YES / NO)

Frame (Frame_num)

End_exec

This statement creates the queue, the queue can be readied upon creation by specifying Ready or Not_ready keyword in the status parm. Frame num will be ignored in Program file, since the system knows the frame in which the statement is running, via the Run_id; In Resource definition file the system lets the user specify the frame in which the queue has to be created; and FRAME_NUM_NULL implies all frames.

13) Exec RCP [Resource definition file/Program file]

Read Queue

Name (queue_name)

Elem (value)

Pointer (data_pointer)

Error_Check (YES / NO)

Frame (Frame_num)

End_exec

This statement is used to read the elements of a queue.

14) Exec RCP [Resource definition file/Program file]

Add Queue

Name (queue_name)

Elem_size (value)

Pointer (data_pointer)

Status (Ready / Not_Ready)

Error_Check (YES / NO)

Frame (Frame_num)

End_exec

This statement is used to add an element to queue. The return value, if greater than zero, indicates the element number added to the queue.

15) Exec RCP [Resource definition file/Program file]

Create Queue_Array

Name (queue_name)

Num_of_elem (value)

Queue_elem_size (value)

Num_of_queue_elem (value)

Error_Check (YES / NO)

Frame (Frame_num)

End_exec

This statement will create a Queue_Array.

16) Exec RCP [Program file]

Rebind Queues

Error_Check (YES / NO)

End_exec

This statement will rebind the inputs and outputs of the function.

17) Exec RCP [Program file]

Release Queues

Error_Check (YES / NO)

End_exec

This statement will release the input and output bindings of the function. Note that the function should be dependent upon a Rcp_Gate.